A Statistical Learning Theory Approach of Bloat

Olivier Teytaud, Sylvain Gelly, Nicolas Bredeche, Marc Schoenauer

A Statistical Learning Theory Approach of Bloat

Olivier Teytaud, Sylvain Gelly, Nicolas Bredeche, Marc Schoenauer
Equipe TAO - INRIA Futurs
LRI, Bat. 490,
University Paris-Sud
91405 Orsay Cedex. France
{*firstname.name*}*@lri.fr*

Review Category : Genetic Programming
The author confirms that the material in this paper represents substantially new work that has not been previously published by conferences, journals, or edited books in the evolutionary computation field.

ABSTRACT

Code bloat, the excessive increase of code size, is an important issue in Genetic Programming (GP). This paper proposes a theoretical analysis of code bloat in the framework of symbolic regression in GP, from the viewpoint of Statistical Learning Theory, a well grounded mathematical toolbox for Machine Learning. Two kinds of bloat must be distinguished in that context, depending whether the target function lies in the search space or not. Then, important mathematical results are proved using classical results from Statistical Learning. Namely, the Vapnik-Cervonenkis dimension of programs is computed, and further results from Statistical Learning allow to prove that a parsimonious fitness ensures Universal Consistency (the solution minimizing the empirical error does converge to the best possible error when the number of samples goes to infinity). However, it is proved that the standard method consisting in choosing a maximal program size depending on the number of samples might still result in programs of infinitely increasing size whith their accuracy; a more complicated modification of the fitness is proposed that theoretically avoids unnecessary bloat while nevertheless preserving the Universal Consistency.

1. INTRODUCTION

Code bloat (or code growth) denotes the growth of program size during the course of Genetic Programming (GP) runs. It has been identified as a key problem in GP from the very beginning [5], and to any variable length representations based learning algorithm [7]. It is today a well studied phenomenon, and empirical solutions have been proposed to effectively address the issue of code bloat (see section 2). However, very few theoretical studies have addressed the issue of bloat.

The purpose of this paper is to provide some theoretical insights into the bloat phenomenon, in the context of symbolic regression by GP, from the Statistical Learning Theory viewpoint[19]. Indeed, Statistical Learning Theory is a recent, yet mature, area of Machine Learning that provides efficient theoretical tools to analyse aspects of learning accuracy and algorithm complexity. Our goal is both to perform an in-depth analysis of bloat and to provide, if possible, appropriate theoretical solutions to avoid it.

The paper is organized as follows : in section 2 we briefly survey some explanations for code bloat that have been proposed in the litterature. Section 3 sets the scenery, and provides an informal description of our results from a GP perspective before discussing their interest for the GP practitioner. Section 4 gives a brief overview of the basic results of Learning Theory that will be used in Section 5 to formally prove all the advertised results. Finally, section 6 discusses the consequences of those theoretical results for GP practitioners and gives some perspectives about this work.

2. CODE BLOAT IN GP

There exists several theories that intend to explain code bloat :

- the *introns* theory states that code bloat acts as a protective mechanism in order to avoid the destructive effects of operators once relevant solutions have been is found[14, 13, 2]. Introns are pieces of code that have no influence on the fitness: either sub-programs that are never executed, or sub-programs who have no effect;
- the *fitness causes bloat* theory relies on the assumption that there is a greater probability to find a bigger program with the same behavior (i.e. semantically equivalent) than to find a shorter one. Thus, once a good solution is found, programs naturally tends to grow because of fitness pressure[9]. This theory states that code bloat is operator-independent and may happen for any variable length representation-based algorithm. As a consequence, code bloat is not to be limited to population-based stochastic algorithm (such as GP), but may be extended to many algorithms using variable length representation [7];
- the *removal bias* theory states that removing longer subprograms is more tacky than removing shorter ones (because of possible destructive consequence), so there is a natural bias that benefits to the preservation of longer programs[17].

While it is now considered that each of these theories somewhat captures part of the problem[1], there has not been any definitive global explanation of the bloat phenomenon. At the same time,

no definitive practical solution has been proposed that would avoid the drawbacks of bloat (increasing evaluation time of large trees) while maintaining the good performances of GP on difficult problems. Some common solutions rely either on specific operators (e.g. size-fair crossover [8], or different Fair Mutation [10]), on some parsimony-based penalization of the fitness [18] or on abrupt limitation of the program size such as the one originally used by Koza[5]. Some other more particular solutions have been proposed but are not widely used yet [15, 16, 11].

3. CONTEXT AND MAIN RESULTS

In this paper, we intend to use Statistical Learning Theory to study code bloat, and to try to help designing algorithm that do not suffer from excessive code bloat, if at all possible.
However, the main goal of Statistical Learning Theory is to study the convergence of Learning algorithms for Machine Learning problems with respect to the number of available samples and the complexity of the hypothesis space. In the framework of this work – symbolic regression using GP – such results amount to study the algorithms with respect to the number of fitness cases and the allowed size of the GP trees.

3.1 Universal Consistency

In this paper, we intend to prove, under some sufficient conditions, that the solution given by GP actually converges, when the number of examples goes to infinity, toward the actual function used to generate the examples. This property is known in Statistical Learning as **Universal Consistency**. Note that this notion is a slightly different from that of Universal Approximation, that people usually refer to when doing symbolic regression in GP: because polynomial for instance are known to be able to approximate any continuous function, GP search using operators $\{+, *\}$ is also assumed to be able to approximate any continuous function. However, Universal Consistency is concerned with the behavior of the algorithm when the number of samples goes to infinity: being able to find a polynomial that approximate a given function at any arbitrary precision does not imply that any interpolation polynomial built from an arbitrary set of sample points will converge to that given function when the number of points goes to infinity.

But going back to bloat, and sticking to the polynomial example, it is also clear that the degree of the interpolation polynomial of a set of samples increases linearly with the number of samples. This leads us to start our bloat analysis by defining two kinds of bloat.

3.2 Structural vs functional bloat

On the one hand, we define the **structural bloat** as the code bloat that unavoidably takes place when at least one optimal solution (a function that exactly matches all possible samples) does not lie in the search space. In such a situation, optimal solutions of increasing accuracy will also exhibit an increasing complexity, as larger and larger code will be generated in order to better approximate the target function.
The extreme case of structural bloat has also been demonstrated in [4]. The authors use some polynomial functions of increasing difficulty, and demonstrate that a precise fit can only be obtained through an increased bloat (see also [3] for related issues about problem complexity in GP).

On the other hand, we define the **functional bloat** as the bloat that takes place when programs length keep on growing even though an optimal solution (of known complexity) does lie in the search space. In order to clarify this point, let us use a simple symbolic regression problem defined as follow : given a set $\mathcal{S}$ of *examples*, the goal is to find a function f (here, a GP-tree) that minimized the Least Square Error (or LSE). If we intend to approximate a polynomial (ex. : $14 * x^2$), we may observe code bloat since it is possible to find arbitrarily long polynoms that gives the exact solution (ex. : $14 * x^2 + 0 * x^3 + ...$). Most of the works cited in section 2 were in fact studying functional bloat: Functional bloat is indeed the most simple, yet already problematic, kind of bloat.

3.3 Overview of results

In section 5, we shall investigate the Universal Consistency of Genetic Programming algorithm, and study in detail structural and functional bloat that might take place when searching program spaces using GP.
A formal and detailed definition of the program space that will be assumed for GP is given in Lemma 1, section 5, and two types of results will then be derived:

- *Universal Consistency* results, i.e. does the probability of misclassification of the solution given by GP converges to the optimal probability of misclassification when the number of examples goes to infinity?
- Bloat-related results, first regarding structural bloat, that will be proved to be incompatible with accuracy, and second with respect to functional bloat, for which the consequences of introducing various types of fitness penalization and/or bound on the complexity of the programs on the behavior of the complexity of the solution will be thoroughly studied.

Let us now state precisely, yet informally, our main results:
– First, as already mentioned, we will precisely define the set of programs under examination, and prove that such a search space fulfills the conditions of the standard theorems of Statistical Learning Theory listed in Section 4.
– Applying those theorems will immediately lead to a first Universal Consistency result for GP, provided that some penalization for complexity is added to the fitness (Theorem 3)
– The first bloat-related result, Proposition 4, unsurprisingly proves that if the optimal function does not belong to the search space, then converging to the optimal error implies that the complexity of the empirical optimal solution goes to infinity (unavoidable structural bloat).
– Theorem 5 is also a negative result about bloat, as it proves that even if the optimal function belongs to the search space, minimizing the LSE alone might lead to (structural) bloat (i.e. the complexity of the empirical solutions goes to infinity with the sample size).
– But the last two theorems (5' and 6) are the best positive results one could expect considering the previous findings: it is possible to carefully adjust the parsimony pressure so as to obtain both Universal Consistency and bounds on the complexity of the empirical solution (i.e. no bloat).

Note that, though all proofs in Section 5 will be stated and proved in the context of classification (i.e. find a function from $\mathbb{R}^d$ into $\{0, 1\}$), their generalization to regression (i.e. find a function from $\mathbb{R}^d$ into $\mathbb{R}$) is straightforward.

3.4 Discussion

First of all, it is important to note that all those results in fact study the solution given by perfectly successful GP runs on the search

space at hand: given a set of samples and a fitness function based on the the Least Square Error (and possibly including some parsimony penalization), it will be assumed that GP does find one program in that search space that globally minimizes this fitness — and it is the behavior of this ideal solution when the number of examples goes to infinity that is theoretically studied.

Or course, we all know that GP is not such an ideal search procedure, and hence such results might look rather far away from GP practice, where the user desperately tries to find a program that gives a reasonably low empirical approximation error. Nevertheless, Universal Consistency is vital for the practitioner too: indeed, it would be totally pointless to fight to approximate an empirically optimal function without any guarantee that this empirical optimum is anywhere close to the ideal optimal solution we are in fact looking for.

Furthermore, the bloat-related results give some useful hints about the type of parsimony that has a chance to efficiently fight the unwanted bloat, while maintaining the Universal Consistency property – though some actual experiments will have to be run to confirm the usefulness of those theoretical hints.

4. ELEMENTS OF LEARNING THEORY

In the frameworks of regression and classification, Statistical Learning Theory [19] is concerned with giving some bounds on the generalization error (i.e. the error on yet unseen data points) in terms of the actual empirical error (the LSE error above) and some fixed quantity depending only on the search space. More precisely, we will use here the notion of *Vapnik-Cervonenkis dimension* (in short, VCdim) of a function space, that somehow gives bounds on the variance of possible better solutions of the regression problem than the one obtained from the limited set of examples.

Consider a set of s examples $(x_i, y_i)_{i\in\{1,\dots,s\}}$. These examples are drawn from a distribution P on the couple (X, Y). They are independent identically distributed, $Y = \{0, 1\}$ (classification problem), and typically $X = \mathbb{R}^d$ for some dimension d.
For any function f, define the *loss* $L(f)$ to be the expectation of $|f(X) - Y|$. Similarly, define the *empirical loss* $\hat{L}(f)$ as the loss observed on the samples: $\hat{L}(f) = \frac{1}{s}\sum_i |f(x_i) - y_i|$.
Finally, define L^*, the *Bayes error*, as the smallest possible generalization error for any mapping from X to $\{0, 1\}$.

The following 4 theorems are well-known in the Statistical Learning community:

Theorem A [6, Th. 12.8, p206] :
Consider $\mathcal{F}$ a family of functions from a domain X to $\{0, 1\}$ and V its VC-dimension. Then, for any $\epsilon > 0$

$$P(\sup_{P\in\mathcal{F}} |L(P) - \hat{L}(P)| \geq \epsilon) \leq 4exp(4\epsilon + 4\epsilon^2)s^{2V}exp(-2s\epsilon^2)$$

and for any $\delta \in]0, 1]$

$$P(\sup_{P\in\mathcal{F}} |L(P) - \hat{L}(P)| \geq \epsilon(s, V, \delta)) \leq \delta$$

where $\epsilon(s, V, \delta) = \sqrt{\frac{4-\log(\delta/(4s^{2V}))}{2s-4}}$.

Other forms of this theorem have no $\log(n)$ factor ; they are known as Alexander's bound, but the constant is so large that this result is not better than the result above unless s is huge ([6, p207]): if $s \geq 64/\epsilon^2$,

$$P(\sup_{P\in\mathcal{F}} |L(P) - \hat{L}(P)| \geq \epsilon) \leq 16(\sqrt{s}\epsilon)^{4096V}exp(-2s\epsilon^2)$$

We classically derive the following result from theorem A:

Theorem A' :
Consider $\mathcal{F}_s$ for $s \geq 0$ a family of functions from a domain X to $\{0, 1\}$ and V_s its VC-dimension. Then,
$\sup_{P\in\mathcal{F}_s} |L(P) - \hat{L}(P)| \to 0$ as $s \to \infty$
almost surely whenever $V_s = o(s/\log(s))$.

Proof :
We use the classical Borell-Cantelli lemma[1], for any $\epsilon \in [0, 1]$:

$$\sum_{s\geq 64/\epsilon^2} P(|L(P)-\hat{L}(P)| > \epsilon) \leq 16 \sum_{s\geq 64/\epsilon^2} (\sqrt{s}\epsilon)^{4096V_s}exp(-2s\epsilon^2)$$

$$\leq 16 \sum_{s\geq 64/\epsilon^2} exp(4096V_s(\log(\sqrt{s}) + \log(\epsilon)) - 2s\epsilon^2)$$

which is finite as soon as $V_s = o(s/\log(s))$.

Theorem B in [6, Th. 18.2, p290] :
Let $\mathcal{F}_1, \dots, \mathcal{F}_k \dots$ with finite VC-dimensions $V_1, \dots, V_k, \dots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Then, being given a sample of size s, consider $\hat{P} \in \mathcal{F}_s$ minimizing the empirical risk $\hat{L}$ among F_s.
Then, if $V_s = o(s/log(s))$ and $V_s \to \infty$,

$$P(L(\hat{P}) \leq \hat{L}(\hat{P}) + \epsilon(s, V_s, \delta)) \geq 1 - \delta$$

$$P(L(\hat{P}) \leq \inf_{P\in\mathcal{F}_s} L(P) + 2\epsilon(s, V_s, \delta)) \geq 1 - \delta$$

$$\text{and } L(\hat{P}) \to \inf_{P\in\mathcal{F}} L(P) \text{ a.s.}$$

Note that for a well chosen family of functions (typically, programs), $\inf_{P\in\mathcal{F}} L(P) = L^*$ for any distribution.

Theorem C (8.14 and 8.4 in [12]) :
Let $H = \{x \mapsto h(a, x); a \in R^{d'}\}$ where h can be computed with at most t' operations among

- $\alpha \mapsto \exp(\alpha)$;
- $+, -, \times, /$;
- jumps conditioned on $>, \geq, =, \leq, =$;
- output 0 ;
- output 1.

Then $VCdim(H) \leq t'^2 d'(d' + 19\log_2(9d'))$.

Furthermore, if $exp(.)$ is used at most q' times, and if there are at most t' operations executed among arithmetic operators, conditional jumps, exponentials,

$$\pi(H, m) \leq 2^{(d'(q'+1))2/2}(9d'(q'+1)2^t)^{5d'(q'+1)}(em(2^{t'}-2)/d')^{d'}$$

where $\pi(H, m)$ is the m^{th} shattering coefficient of H, and hence

$$VCdim(H) \leq (d'(q'+1))^2 + 11d'(q'+1)(t' + \log_2(9d'(q'+1)))$$

[1] If $\sum_n P(X_n > \epsilon)$ is finite for any $\epsilon > 0$ and $X_n > 0$, then $X_n \to 0$ almost surely.

Finally, if $q = 0$ then $VCdim(H) \leq 4d'(t' + 2)$.

Theorem D : structural risk minimization, [6] p. 294
Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ with finite VC-dimensions $V_1, \ldots, V_k, \ldots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Assume that all distribution lead to $L_{\mathcal{F}} = L^*$ where L^* is the optimal possible error. Then, given a sample of size s, consider $f \in \mathcal{F}$ minimizing $\hat{L}(f) + \sqrt{\frac{32}{s} V(f) \log(e \times s)}$, where $V(f)$ is V_k with k minimal such that $f \in \mathcal{F}_k$. Then :

• if additionally one optimal function belongs to $\mathcal{F}_k$,
then for any s and ϵ such that $V_k \log(e \times s) \leq s\epsilon^2/512$,
generalization error is lower than ϵ with probability at most
$\Delta \exp(-s\epsilon^2/128) + 8s^{V_k} \times \exp(-s\epsilon^2/512)$
where $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is assumed finite.

• the generalization error, with probability 1, converges to L^*.

5. RESULTS

This section presents in details results that have been already surveyed in Section 3. They make an intensive use of the results of Statistical Learning Theory presented in the previous section.
More precisely, Lemma 1 and Lemma 1' define precisely the space of program considered here, and carefully show that it satisfies the hypotheses of Theorems A-C of section 3. This allows us to evaluate the VC-dimension of sets of programs, stated in Theorem 2.
Theorem 3 is the general result that guarantees the Universal Consistency for Genetic Programming through complexity penalization by applying Theorem D of section 3.
Proposition 4 straightforwardly and unsurprisingly shows that if the optimal function does not lie in the considered space of programs, then converging toward the optimal generalization error implies that the complexity of the solution goes to infinity (*structural bloat*). Theorem 5, another negative result about bloat, and shows that a relevant choice of VC-dimension depending upon the sample size ensures Universal Consistency, but leads (for some distribution of examples) to divergence of the program complexity to infinity (i.e. *code bloat*), whenever there exists a very simple optimal program.
Theorem 5' shows that, using a relevant a priori bound on the complexity of the program and adding a user-defined complexity penalization to the fitness, can lead to convergence toward a user-defined compromise between classification rate and program complexity (i.e. we ensure almost sure convergence to a compromise of the form "λ_1 CPU time + λ_2 misclassification rate + λ_3 number of lines", where the λ_i are user-defined.
Finally, next we propose a new approach combining an a priori limit on VC-dimension (i.e. *size limit*) and a complexity penalization (i.e. *parsimony pressure*) and state in theorem 6 that this leads to both universal consistency and convergence to an optimal complexity of the program (i.e. *no-bloat*).

Lemma 1 :

Let F be the set of functions which can be computed with at most t operations among :

- operations $\alpha \mapsto \exp(\alpha)$ (at most q times);
- operations $+, -, \times, /$;
- jumps conditioned on $>, \geq, =, \leq, =$;

and

- output 0 ;
- output 1 ;
- labels for jumps ;
- at mosts m constants ;
- at most z variables

by a program with at most n lines.

We note $\log_2(x)$ the integer part (ceil) of $\log(x)/\log(2)$. Then F is included in H as defined above, for a given P with $t' = t + t\ max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + max(9log_2(z), 0) + max(3log_2(z) - 3, 0))$, $q' = q$, $d' = 1 + m$.

Proof :
We define a program as in theorem above that can emulate any of these programs, with at most $t' = t + t\ max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + max(9log_2(z), 0) + max(3log_2(z) - 3, 0))$, $q' = q$, $d' = 1 + m$.
The program is as follows :

- label "inputs"
- initialize $variable(1)$ at value $x(1)$
- initialize $variable(2)$ at value $x(2)$
- $\ldots$
- initialize $variable(dim(x))$ at value $x(dim(x))$
- label "constants"
- initialize $variable(dim(x) + 1)$ at value a_1
- initialize $variable(dim(x) + 2)$ at value a_2
- $\ldots$
- initialize $variable(dim(x) + m)$ at value a_m
- label "Decode the program into c"
- operation decode c
- label "Line 1"
- operation $c(1, 1)$ with variables $c(1, 2)$ and $c(1, 3)$ and $c(1, 4)$
- label "Line 2"
- operation $c(2, 1)$ with variables $c(2, 2)$ and $c(2, 3)$ and $c(2, 4)$
- $\ldots$
- label "Line n"
- operation $c(n, 1)$ with variables $c(n, 2)$and $c(n, 3)$ and $c(n, 4)$
- label "output 0"
- output 0
- label "output 1"
- output 1

"operation decode c" can be developed as follow. Indeed, we need m real numbers, for parameters, and $4n$ integers $c(.,.)$, that we will encode as only one real number in $[0, 1]$ as follows :

1. let $y \in [0, 1]$
2. for each $i \in [1, \ldots n]$:
   - $c(i, 1) = 0$
   - $y = y * 2$
   - if $(y > 1)$ then $\{\ c(i, 1) = 1\ ;\ y = y - 1\ \}$
   - $y = y * 2$
   - if $(y > 1)$ then $\{\ c(i, 1) = c(i, 1) + 2\ ;\ y = y - 1\ \}$
   - $y = y * 2$
   - if $(y > 1)$ then $\{\ c(i, 1) = c(i, 1) + 4\ ;\ y = y - 1\ \}$
3. for each $j \in [2, 4]$ and $i \in [1, \ldots n]$:
   - $c(i, j) = 0$
   - $y = y * 2$
   - if $(y > 1)$ then $\{\ c(i, j) = 1\ ;\ y = y - 1\ \}$
   - $y = y * 2$
   - if $(y > 1)$ then $\{\ c(i, j) = c(i, j) + 2\ ;\ y = y - 1\ \}$

- $y = y * 2$
- if $(y > 1)$ then $\{ c(i,j) = c(i,j) + 4 \; ; \; y = y - 1 \}$
- ...
- $y = y * 2$
- if $(y > 1)$ then $\{ c(i,j) = c(i,j) + 2^{log_2(z)-1} \; ; \; y = y - 1 \}$

The cost of this is $n \times (3 + max(3 \times log_2(z), 0))$ "if then", and $n \times (3 + max(3 \times log_2(z), 0))$ operators $\times$, and $n(2 + max(3(log_2(z) - 1), 0))$ operators $+$, and $n \times (3 + max(3 \times log_2(z), 0))$ operators $-$. The overall sum is bounded by $n(11 + max(9 \log_2(z), 0) + max(3log_2(z) - 3, 0))$.

Lemma 1' : "operation $c(i,1)$ with variables c(i,2) and c(i,3)" can be developed as follows:

- if $c(i,1) == 0$ then goto "output1"
- if $c(i,1) == 1$ then goto "output 0"
- if $c(i,2) == 1$ then $c = variable(1)$
- if $c(i,2) == 2$ then $c = variable(2)$
- ...
- if $c(i,2) == z$ then $c = variable(z)$
- if $c(i,1) == 7$ then goto "Line c" (must be encoded by dichotomy with $log_2(n)$ lines)
- if $c(i,1) == 6$ then goto "exponential(i)"
- if $c(i,3) == 1$ then $b = variable(1)$
- if $c(i,3) == 2$ then $b = variable(2)$
- ...
- if $c(i,3) == z$ then $b = variable(z)$
- if $c(i,1) == 2$ then $a = c + b$
- if $c(i,1) == 3$ then $a = c - b$
- if $c(i,1) == 4$ then $a = c \times b$
- if $c(i,1) == 5$ then $a = c/b$
- if $c(i,4) == 1$ then $variable(1) = a$
- if $c(i,4) == 2$ then $variable(2) = a$
- ...
- if $c(i,4) == z$ then $variable(z) = a$
- label "endOfInstruction(i)"

For each such instruction, at the end of the program, we add three lines of the following form :

- label "exponential(i)"
- $a = exp(c)$
- goto "endOfInstruction(i)"

Each sequence of the form "if x=... then" (p times) can be encoded by dichotomy with $log_2(p)$ tests "if ... then goto".

Theorem 2 :
Let F be the set of programs as in lemma 1, where $q' \geq q$, $t' \geq t + t \; max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + max(9log_2(z), 0) + max(3log_2(z) - 3, 0))$, $d' \geq 1 + m$.

$$VCdim(H) \leq t'^2 d'(d' + 19 \log_2(9d'))$$

$$VCdim(H) \leq (d'(q'+1))^2 + 11d'(q'+1)(t' + \log_2(9d'(q'+1)))$$

If $q = 0$ (no exponential) then $VCdim(H) \leq 4d'(t' + 2)$.

Proof : Just plug Lemmas 1 and 1' in Theorem C ■

Theorem 3 :
Consider q_f, t_f, m_f, n_f and z_f integer sequences, non-decreasing functions of f. Define $V_f = VCdim(H_f)$, where H_f is the set of programs with at most t_f lines executed, with z_f variables, n_f lines, q_f exponentials, and m_f constants.

Then with ${q'}_f = q_f$, ${t'}_f = t_f + t_f \; max(3 + \log_2(n_f) + \log_2(z_f), 7 + 3\log_2(z_f)) + n_f(11 + max(9log_2(z_f), 0) + max(3log_2(z_f) - 3, 0))$, ${d'}_f = 1 + m_f$,

$$V_f = ({d'}_f({q'}_f+1))^2 + 11{d'}_f({q'}_f+1)({t'}_f + \log_2(9{d'}_f({q'}_f+1)))$$

or, if $\forall f \; q_f = 0$ then define $V_f = 4{d'}_f({t'}_f + 2)$.

Then, being given a sample of size s, consider $f \in \mathcal{F}$ minimizing $\hat{L}(f) + \sqrt{\frac{32}{s} V(f) \log(e \times s)}$, where $V(f)$ is the infimum of all k such that $f \in \mathcal{F}_k$.

Then, if $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is finite,

- the generalization error, with probability 1, converges to L^*.
- if one optimal rule belongs to $\mathcal{F}_k$, then for any s and ϵ such that $V_k \log(e \times s) \leq s\epsilon^2/512$, the generalization error is lower than ϵ with probability at most $\Delta \exp(-s\epsilon^2/128) + 8s^{V_k} \times \exp(-s\epsilon^2/512)$ where $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is assumed finite.

Proof : Just plug theorem D in theorem 2. ■

We now prove the non-surprising fact that if it is possible to approximate the optimal function (the Bayesian classifier) without reaching it exactly, then the "complexity" of the program runs to infinity as soon as there is convergence of the generalization error to the optimal one.

Proposition 4:
Consider P_s a sequence of functions such that $P_s \in \mathcal{F}_{V(s)}$, with $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \ldots$, where $\mathcal{F}_V$ is a set of functions from X to $\{0,1\}$ with VC-dimension bounded by V.
Define $L_V = \inf_{P \in \mathcal{F}_V} L(P)$ and $V(P) = \inf\{V / P \in \mathcal{F}_V\}$
and suppose that $\forall V \; L_V > L^*$.

Then

$$(L(P_s) \stackrel{s \to \infty}{\longrightarrow} L^*) \Longrightarrow (V(P_s) \stackrel{s \to \infty}{\longrightarrow} \infty)$$

Proof:
Define $\epsilon(V) = L_V - L^*$. Assume that $\forall V \; \epsilon(V) > 0$. ϵ is necessarily non-increasing.

Consider V_0 a positive integer ; let us prove that if n is large enough, then $V(P_s) \geq V_0$.

There exists ϵ_0 such that $\epsilon(V_0) > \epsilon_0 > 0$.

For s large enough, $L(P_s) \leq L^* + \epsilon_0$,
hence $L_{V_s} \leq L^* + \epsilon_0$,
hence $L^* + \epsilon(V_s) \leq L^* + \epsilon_0$,
hence $\epsilon(V_s) \leq \epsilon_0$,
hence $V_s > V_0$. ■

We now show that the usual procedure defined below, consisting in defining a maximum VC-dimension depending upon the sample

size (as usually done in practice and as recommended by theorem B) and then using a moderate family of functions, leads to bloat. With the same hypotheses as in theorem B, we can state

Theorem 5 (bloat theorem for empirical risk minimization with relevant VC-dimension):
Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ non-empty sets of functions with finite VC-dimensions $V_1, \ldots, V_k, \ldots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Then, given a sample of size s, consider $\hat{P} \in \mathcal{F}_s$ minimizing the empirical risk $\hat{L}$ in F_s. From Theorem B we already know that
if $V_s = o(s/log(s))$ and $V_s \to \infty$,
then $P(L(\hat{P}) \leq \hat{L}(\hat{P}) + \epsilon(s, V_s, \delta)) \geq 1 - \delta$,
and $L(\hat{P}) \to \inf_{P \in \mathcal{F}} L(P)$ a.s..

We will now state that if $V_s \to \infty$,
and denotating $V(f) = min\{V_k / f \in \mathcal{F}_k\}$, then

$\forall V_0, P_0 > 0$
$\exists P$, distribution of probability on X and Y, such that
$\exists g \in \mathcal{F}_1$ such that $L(g) = L^*$
and for s sufficiently large $P(V(\hat{P}) \leq V_0) \leq P_0$.

Remarks :
The result in particular implies that for any V_0, there is a distribution of examples such that for some g with $V(g) = V_1$ and $L(g) = L^*$, with probability 1, $V(\hat{f}) \geq V_0$ infinitely often as s increases.

Proof (of the part which is not theorem B) :
Consider $V_0 > 0$ and $P_0 > 0$. Consider α such that $(e\alpha/2^\alpha)^{V_0} \leq P_0/2$. Consider s such that $V_s \geq \alpha V_0$. Let $d = \alpha V_0$.

Consider $x_1, \ldots, x_d$ shattered by $\mathcal{F}_d$.

Define the probability measure P by the fact that X and Y and independent and $P(Y = 1) = \frac{1}{2}$ and $P(X = x_i) = \frac{1}{d}$.

Then, the following holds, with Q the empirical distribution (the average of Dirac masses on the x_i's) :

1. no empty x_i's :
$$P(E_1) \to 0$$
where E_1 is the fact that $\exists i / Q(X = x_i) = 0$, as $s \to \infty$.
2. no equality :
$$P(E_2) \to 0$$
where E_2 is the fact that E_1 occurs or $\exists i / Q(Y = 1 | X = x_i) = \frac{1}{2}$.
3. the best function is not in $\mathcal{F}_{V_0}$:
$$P(E_3 | E_2 \text{ does not hold}) \leq S(d, d/\alpha)/2^d$$
where E_3 is the fact that $\exists g \in \mathcal{F}_{d/\alpha = V_0} / \hat{L}(g) = inf_{\mathcal{F}_d} \hat{L}$, with $S(d, d/\alpha)$ the relevant shattering coefficient, ie the cardinal of $\mathcal{F}_{d/\alpha}$ restricted to $\{x_1, \ldots, x_d\}$.

We now only have to use classical results. It is well known in VC-theory that $S(a, b) \leq (ea/b)^b$ (see for example [6, chap.13]), hence $S(d, d/\alpha) \leq (ed/(d/\alpha))^{d/\alpha}$ and
$$P(E_3 | E_2 \text{ does not hold}) \leq (e\alpha)^{d/\alpha}/2^d \leq P_0/2$$
and if n is sufficiently large to ensure that $P(E_2) \leq P_0/2$ (we have proved above that $P(E_2) \to 0$ as $s \to \infty$) then
$$P(E_3) \leq P(E_3 | \neg E_2) \times P(\neg E_2) + P(E_2)$$
$$\leq P(E_3 | \neg E_2) + P(E_2) \leq P_0/2 + P_0/2 \leq P_0 \quad \blacksquare$$

We now show that, on the other hand, it is possible to optimize a compromise between optimality and complexity in an explicit manner (e.g., replacing 1 % precision with 10 lines of programs or 10 minutes of CPU) :

Theorem 5' (bloat-control theorem for regularized empirical risk minimization with relevant VC-dimension):
Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ be non-empty sets of functions with finite VC-dimensions $V_1, \ldots, V_k, \ldots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Consider W a user-defined complexity penalization term. Then, being given a sample of size s, consider $P \in \mathcal{F}_s$ minimizing the regularized empirical risk $\tilde{\hat{L}}(P) = \hat{L}(P) + W(P)$ among F_s. If $V_s = o(s/log(s))$ and $V_s \to \infty$, then $\tilde{L}(\hat{P}) \to \inf_{P \in \mathcal{F}} \tilde{L}(P)$ a.s. where $\tilde{L}(P) = L(P) + W(P)$.

Proof :
$$\sup_{P \in \mathcal{F}_s} |\tilde{\hat{L}}(P) - \tilde{L}(P)|$$
$$\leq \sup_{P \in \mathcal{F}_s} |\hat{L}(P) - L(P)|$$
$$\leq \epsilon(s, V_s) \to 0 \text{ almost surely, by theorem A'}$$
Hence the expected result. $\blacksquare$

Remark: the drawback of this approach is that we have lost universal consistency and consistency (in the general case, the misclassification rate in generalization will not converge to the Bayes error, and whenever an optimal program exists, we will not necessarily converge to its efficiency).

We now turn our attention to a more complicated case where we want to ensure universal consistency, but we want to avoid a non-necessary bloat ; e.g., we require that if an optimal program exists in our family of functions, then we want to converge to its error rate, without increasing the complexity of the program.

We are now going to consider a merge between regularization and bounding of the VC-dimension ; we penalize the complexity (eg, length) of programs by a penalty term $R(s, P) = R(s)R'(P)$ depending upon the sample size and upon the program ; $R(.,.)$ is user-defined and the algorithm will look for a classifier with a small value of both R' and L.

We study both the universal consistency of this algorithm (ie $L \to L^*$) and the no-bloat theorem (ie $R' \to R'(P^*)$ when P^* exists).

Theorem 6 :
Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ with finite VC-dimensions $V_1, \ldots, V_k, \ldots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Define $V(P) = V_k$ with $k = \inf\{t | P \in \mathcal{F}_t\}$. Define $L_V = inf_{P \in \mathcal{F}_V} L(P)$. Consider $V_s = o(log(s))$ and $V_s \to \infty$. Consider $\hat{P}$ minimizing $\hat{L}(P) + R(s, P)$ in $\mathcal{F}_s$ and assume that $R(s, .) \geq 0$.

Then (consistency), whenever $\sup_{P \in \mathcal{F}_{V_s}} R(s, P) = o(1)$, $L(\hat{P}) \to \inf_{P \in \mathcal{F}} L(P)$ almost surely (note that for well chosen family of functions, $\inf_{P \in \mathcal{F}} L(P) = L^*$)

Assume that $\exists P^* \in \mathcal{F}_{V^*}$ $L(P^*) = L^*$. Then with $R(s, P) = R(s)R'(P)$ and with $R'(s) = \sup_{P \in \mathcal{F}_{V_s}} R'(P)$:

1. **non-asymptotic no-bloat theorem :** $R'(\hat{P}) \leq R'(P^*) + (1/R(s))2\epsilon(s, V_s, \delta)$ with probability at least $1-\delta$ (this result is in particular interesting for $\epsilon(s, V_s, \delta)/R(s) \to 0$, what is possible for usual regularization terms as in theorem D,

2. **almost-sure no-bloat theorem :** if $R(s)s^{(1-\alpha)/2} = O(1)$, then almost surely $R'(\hat{P}) \to R'(P^*)$ and if $R'(P)$ has discrete values (such as the number of instructions in P or many complexity measures for programs) then for s sufficiently large, $R'(\hat{P}) = R'(P^*)$.

3. **convergence rate :** with probability at least $1-\delta$,

$$L(\hat{P}) \leq inf_{P \in \mathcal{F}_{V_s}} L(P) + \underbrace{R(s)R'(s)}_{=o(1) \text{ by hypothesis}} + 2\epsilon(s, V_s, \delta)$$

where $\epsilon(s, V, \delta) = \sqrt{\frac{4 - \log(\delta/(4s^{2V}))}{2s-4}}$ is an upper bound on $\epsilon(s, V) = \sup_{f \in \mathcal{F}_V} |\hat{L}(f) - L(f)|$ (given by theorem A), true with probability at least $1-\delta$.

Remarks : The usual $R(s, P)$ as used in theorem D or theorem 3 provides consistency and non-asymptotic no-bloat. A stronger regularization leads to the same results, plus almost sure no-bloat. The asymptotic convergence rate depends upon the regularization. The result is not limited to genetic programming and could be used in other areas.
As shown in proposition 4, the no-bloat results require the fact that $\exists V^* \exists P^* \in \mathcal{F}_{V^*}\ L(P^*) = L^*$.
Interestingly, the convergence rate is reduced when the regularization is increased in order to get the almost sure no-bloat theorem.

Proof :
Define $\epsilon(s, V) = \sup_{f \in \mathcal{F}_V} |\hat{L}(f) - L(f)|$.
Let us prove the consistency: For any P,

$$\hat{L}(\hat{P}) + R(s, \hat{P}) \leq \hat{L}(P) + R(s, P)$$

On the other hand,

$$L(\hat{P}) \leq \hat{L}(\hat{P}) + \epsilon(s, V_s)$$

So :

$$L(\hat{P}) \leq (inf_{P \in \mathcal{F}_{V_s}} (\hat{L}(P) + R(s, P))) - R(s, \hat{P}) + \epsilon(s, V_s)$$

$$\leq (inf_{P \in \mathcal{F}_{V_s}} (L(P) + \epsilon(s, V_s) + R(s, P))) - R(s, \hat{P}) + \epsilon(s, V_s)$$

$$\leq (inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s, P))) + 2\epsilon(s, V_s)$$

as $\epsilon(s, V_s) \to 0$ almost surely[2] and $(inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s, P))) \to \inf_{P \in \mathcal{F}} L(P)$, we conclude that $L(\hat{P}) \to \inf_{P \in \mathcal{F}} L(P)$ a.s.

We now focus on the proof of the "no bloat" result :
By definition of the algorithm, for s sufficiently large to ensure $P^* \in \mathcal{F}_{V_s}$,

$$\hat{L}(\hat{P}) + R(s, \hat{P}) \leq \hat{L}(P^*) + R(s, P^*)$$

hence with probability at least $1-\delta$,

[2]See theorem A'

$$R'(\hat{P}) \leq R'(P^*) + (1/R(s))(L^* + \epsilon(s, V_s, \delta) - L(\hat{P}) + \epsilon(s, V_s, \delta))$$

hence

$$R'(\hat{P}) \leq R'(V^*) + (1/R(s))(L^* - L(\hat{P}) + 2\epsilon(s, V_s, \delta))$$

As $L^* \leq L(\hat{P})$, this leads to the non-asymptotic version of the no-bloat theorem.

The almost sure no-bloat theorem is derived as follows.

$$R'(\hat{P}) \leq R'(P^*) + 1/R(s)(L^* + \epsilon(s, V_s) - L(\hat{P}) + \epsilon(s, V_s))$$

hence

$$R'(\hat{P}) \leq R'(P^*) + 1/R(s)(L^* - L(\hat{P}) + 2\epsilon(s, V_s))$$

$$R'(\hat{P}) \leq R'(P^*) + 1/R(s)2\epsilon(s, V_s)$$

All we need is the fact that $\epsilon(s, V_s)/R(s) \to 0$ a.s.

For any $\epsilon > 0$, we consider the probability of $\epsilon(s, V_s)/R(s) > \epsilon$, and we sum over $s > 0$. By the Borel-Cantelli lemma, the finiteness of this sum is sufficient for the almost sure convergence to 0.

The probability of $\epsilon(s, V_s)/R(s) > \epsilon$ is the probability of $\epsilon(s, V_s) > \epsilon R(s)$. By theorem A, this is bounded above by $O(exp(2V_s \log(s) - 2s\epsilon^2 R(s)^2))$. This has finite sum for $R(s) = \Omega(s^{-(1-\alpha)/2})$.

Let us now consider the convergence rate. Consider s sufficiently large to ensure $L_{V_s} = L^*$. As shown above during the proof of the consistency,

$$L(\hat{P}) \leq (inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s, P))) + 2\epsilon(s, V_s)$$

$$\leq (inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s)R'(P))) + 2\epsilon(s, V_s)$$

$$\leq inf_{P \in \mathcal{F}_{V_s}} L(P) + R(s)R'(s) + 2\epsilon(s, V_s)$$

so with probability at least $1-\delta$,

$$\leq inf_{P \in \mathcal{F}_{V_s}} L(P) + R(s)R'(s) + 2\epsilon(s, V_s, \delta) \quad \blacksquare$$

6. CONCLUSION

In this paper, we have proposed a theoretical study of an important issue in Genetic Programming known as code bloat. We have shown that GP trees used in symbolic regression (involving the four arithmetic operations, the exponential function, and ephemeral constants, as well as test and jump instructions) could be applied some classical results from Statistical Learning Theory. This has lead to two kinds of original outcomes: some results about Universal Consistency of GP, i.e. some guarantee that if GP converges to some (empirical) function, this function will be close from the optimal one if sufficiently enough examples are used; and results about the bloat, both the unavoidable structural bloat in case the target ideal function is not included in the search space, and the functional bloat, for which we proved that it can – theoretically – be avoided by simultaneously bounding the length of the programs with some *ad hoc* bound) and using some parsimony pressure in the fitness function. Some negative results have been obtained, too, such as the fact though structural bloat was know to be unavoidable, functional bloat might indeed happen even when the target

function does lie in the search space, but no parsimony pressure is used.

Interestingly enough, all those results (both positive and negative) about bloat are also valid in different contexts, such as for instance that of Neural Networks (the number of neurons replaces the complexity of GP programs). Moreover, results presented here are not limited to the scope of regression problems, but may be applied to variable length representation algorithms in different contexts such as control or identification tasks.

Further research will first be concerned with experimental validations of those theoretical results, emphasizing their usefulness for practitioners (see the discussion Section 3.4). However, we are aware that the balance between both parsimony factors (the bound on the complexity of the search space, and the penalization factor in the fitness) might be tricky to tune, and the solution might prove to be highly problem-dependent.
Another extension of those results concerns noisy and dynamic fitnesses: most of those results could probably be easily generalized to ϵ-convergence instead of actual convergence.
Finally, going back to the debate about the causes of bloat in practice, it is clear that our results can only partly explain the actual cause of bloat in a real GP run – and tends to give arguments to the "fitness causes bloat" explanation [9]. It might be possible to study the impact of size-preserving mechanisms (e.g. specific variation operators, like size-fair crossover [8] or fair mutations [10]) as somehow contributing to the regularization term in our final result ensuring both Universal Consistency and no-bloat.

7. REFERENCES

[1] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.

[2] T. Blickle and L. Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms Workshop at KI-94*. Max-Planck-Institut für Informatik.

[3] J. M. D. et al. What makes a problem gp-hard? analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines*, 2(2):165 – 191, 2001.

[4] S. Gustafson, A. Ekart, E. Burke, and G. Kendall. Problem difficulty and code growth in Genetic Programming. *Genetic Programming and Evolvable Machines*, 4(3):271–290, 2004.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[6] G. L. L. Devroye, L. Györfi. A probabilistic theory of pattern recognition, springer. 1997.

[7] W. B. Langdon. The evolution of size in variable length representations. In *ICEC'98*, pages 633–638. IEEE Press, 1998.

[8] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming And Evolvable Machines*, 1(1/2):95–119, 2000.

[9] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In J. Koza, editor, *Late Breaking Papers at GP'97*, pages 132–140, 1997.

[10] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. Angeline, editors, *Advances in Genetic Programming III*, pages 163–190. MIT Press, 1999.

[11] S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. L. et al., editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, 2002.

[12] P. B. M. Antony. Neural network learning : Theoretical foundations, cambridge university press. 1999.

[13] N. F. McPhee and J. D. Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.

[14] P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[15] A. Ratle and S. M. Avoiding the bloat with probabilistic grammar-guided genetic programming. In P. C. et al., editor, *Artificial Evolution VI*. Springer Verlag, 2001.

[16] S. Silva and J. Almeida. Dynamic maximum tree depth : A simple technique for avoiding bloat in tree-based gp. In E. C.-P. et al., editor, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1776–1787. Springer-Verlag, 2003.

[17] T. Soule. Exons and code growth in genetic programming. In J. A. F. et al., editor, *EuroGP 2002*, volume 2278 of *LNCS*, pages 142–151. Springer-Verlag, 2002.

[18] T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, 1998.

[19] V. Vapnik. The nature of statistical learning, springer. 1995.